ICIC Express Letters
Part B: Applications
Volume **15**, Number **7**, July **2024**

ICIC International ©2024 ISSN 2185-2766
pp. **723–731**

A STUDY ON OPTIMIZATION OF HYPER-PARAMETERS IN DEEP REINFORCEMENT LEARNING BY BAYESIAN OPTIMIZATION

Ryosuke Sota, Takuto Nishimura and Tadashi Horiuchi

Advanced Engineering Faculty
National Institute of Technology, Matsue College
14-4 Nishi-ikuma, Matsue, Shimane 690-8518, Japan
{ s2211, s2311, horiuchi }@matsue-ct.jp

Received November 2023; accepted January 2024

Abstract. *Deep reinforcement learning is a machine learning method that combines deep learning and reinforcement learning. Deep Q-Network (DQN) is one of the typical methods of deep reinforcement learning. DQN uses Convolutional Neural Network (CNN) which can extract features from the input images. We have applied DQN method to the robot navigation problem. The values of hyper-parameters including the network structure of DQN have been determined empirically. In this study, we attempt to optimize the values of hyper-parameters of deep reinforcement learning by using Bayesian optimization. We realized to optimize the values of hyper-parameters including the network structure of DQN by Optuna, a framework of Bayesian optimization. We confirmed that the values of hyper-parameters obtained by Optuna have higher learning performance than those by empirical method.*

Keywords: Deep reinforcement learning, Deep Q-Network (DQN), Bayesian optimization, Tree-structured Parzen Estimator (TPE), Optuna, Mobile robot, Behavior acquisition

1. **Introduction.** Deep reinforcement learning offers new promise for solving complex control tasks using visual information. Developments of Deep Q-Network (DQN) [1] which is superior to human experts in several video games and AlphaGo [2] which won a human champion had strong impact on all over the world. Deep reinforcement learning has remarkable features to directly learn the action policy from the high-dimensional image data by using Convolutional Neural Network (CNN) [3].

In addition, we have already realized that the mobile robot learns to acquire behaviors to reach the goal area avoiding the wall and obstacles with probability of 97.5% in the simulation environment by using DQN-based method [4]. The network structure of the DQN is shown in Figure 1. This network is structured to extract features from camera information using CNN, append 12 units of distance information to the flattened units, and select actions through fully connected layers. The hyper-parameters of this network were empirically determined in our previous research. Here, the hyper-parameters refer to the parameters necessary to be set their values in advance, such as the learning rate, the batch size, and the kind of optimization algorithm used in neural networks.

In this research, we attempt to efficiently optimize the network structure and the values of hyper-parameters of DQN by using Bayesian optimization, for the robot navigation problem to acquire the behaviors to reach the goal area using an ROS-compatible mobile robot equipped with LiDAR and camera. As a result, we realized to optimize the values of hyper-parameters including the network structure of DQN by Optuna, a framework of Bayesian optimization. We found that when using the hyper-parameter values optimized by Optuna, the robot reached the goal area more frequently and with fewer actions

DOI: 10.24507/icicelb.15.07.723

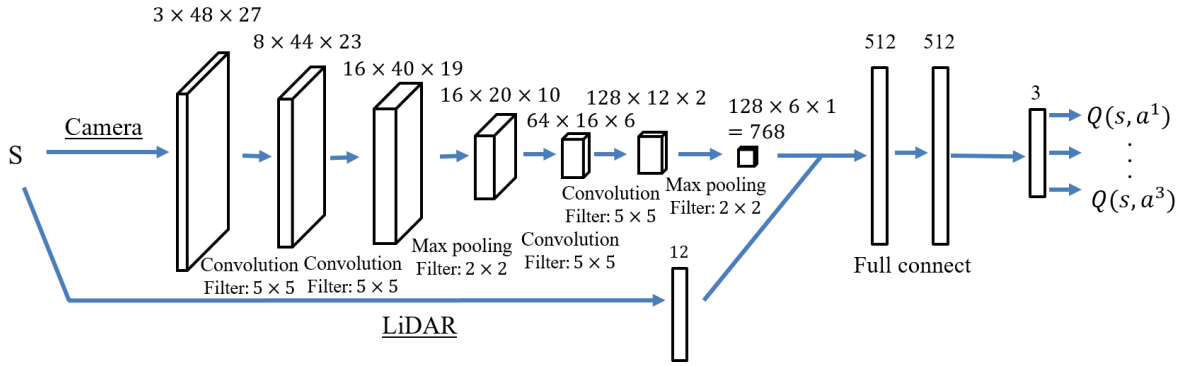


FIGURE 1. DQN network structure determined empirically

compared to when using values optimized by empirical methods. This confirms that the hyper-parameter values optimized by Optuna exhibit higher learning performance.

The paper is organized as follows. Chapter 2 reviews several related works. Chapter 3 provides an explanation of the DQN method and its improvements as applied to robots. Chapter 4 explains parameter optimization using Bayesian optimization. In this study, we utilize the Tree-structured Parzen Estimator (TPE) algorithm, which is one of the techniques in Bayesian optimization. Chapter 5 presents an overview of the features of the TurtleBot3 Burger, the mobile robot under study. Chapter 6 offers a detailed description of the problem setting. This includes defining the problem environment, specifying the definitions of state, action, and reward spaces, and discussing the restart method for cases involving collisions with walls or obstacles. Additionally, information about the optimization targets for Bayesian optimization is provided. Chapter 7 outlines the experimental methods and presents the results. We showcase experiments involving the optimization of hyper-parameters using Bayesian optimization and provide a comparison with parameters optimized empirically. In Chapter 8, we conclude the paper and offer insights into future research directions.

2. **Related Works.**

Kageyama et al. [5] developed a method for detecting obstacles in front of a train using deep learning and a camera to prevent collisions with obstacles on the rails. They further demonstrated that the addition of LiDAR information improved the performance of the detection. Furthermore, the ROS-compatible mobile robot, Turtlebot2, equipped with LiDAR, autonomously learns to avoid collisions with walls using deep reinforcement learning in an indoor environment by utilizing LiDAR's distance information [6].

Ekundayo [7] demonstrated that, in the problem of predicting household electricity consumption, they optimized parameters of advanced CNN methods using Optuna and achieved the best performance compared to several other methods. Uemura et al. [8] are developing a walking navigation system that can automatically detect stairs and steps to inform individuals with visual impairments. As part of this effort, they have utilized a Convolutional Neural Network (CNN) trained on indoor camera data to recognize stairs. The parameters of the CNN have been optimized using Optuna for maximum efficiency. Furthermore, Takaoka et al. [9] demonstrated the optimization of parameters used in advanced reinforcement learning methods with Optuna and applied action learning in a grid world environment.

3. **Method.**

3.1. **Deep Q-Network (DQN).**

DQN uses the Q-learning method [10] which is one of the most widely-used reinforcement learning methods and also uses CNN in order

to approximate the action-value function. The action-value function is shown by the following equation:

$$Q^{\pi}(s,a) = E_{\pi}\left[\sum_{k=0}^{\infty}\gamma^{k}r_{t+k}\middle| s_t = s, a_t = a\right] \tag{1}$$

where π denotes the policy $\pi = P(a|s)$, r_t is reward at time step t, γ is discount rate, s_t represents the state of the environment at time step t, and a_t represents the action chosen by the agent in a given state s_t. E_{π} denotes the expectation following policy π, representing the expected future rewards for (s_t, a_t).

DQN uses the method called experience replay. In order to perform experience replay, the agent's experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ at each time step t are stored in the data set $D = \{e_1, \ldots, e_t\}$. The data set D is also called replay memory. During learning, we apply Q-learning update rule on samples of experience (s_j, a_j, r_j, s_{j+1}) drawn uniformly at random from the data set D.

The Q-learning update at iteration i uses the following Huber loss function:

$$\delta_i(\theta_i) = r_t + \gamma \max_{a'} Q\left(s', a'; \theta_i^{-}\right) - Q(s, a; \theta_i) \tag{2}$$

$$L_i(\delta_i) = \begin{cases} \dfrac{1}{2}\delta_i^2 & \text{if } |\delta_i| < 1 \\ |\delta_i| - \dfrac{1}{2} & \text{otherwise} \end{cases} \tag{3}$$

in which r is the reward, γ is the discount factor, θ_i are the network parameters of the learning network Q at iteration i, and θ_i^{-} are the network parameters of the target network $\hat{Q}$ which is used to compute the target $y_j = r + \gamma \max_{a'} \hat{Q}\left(s', a'; \theta_i^{-}\right)$ at iteration i.

After the learning of action-value function based on this approach during C steps, the target network parameters θ^{-} are updated.

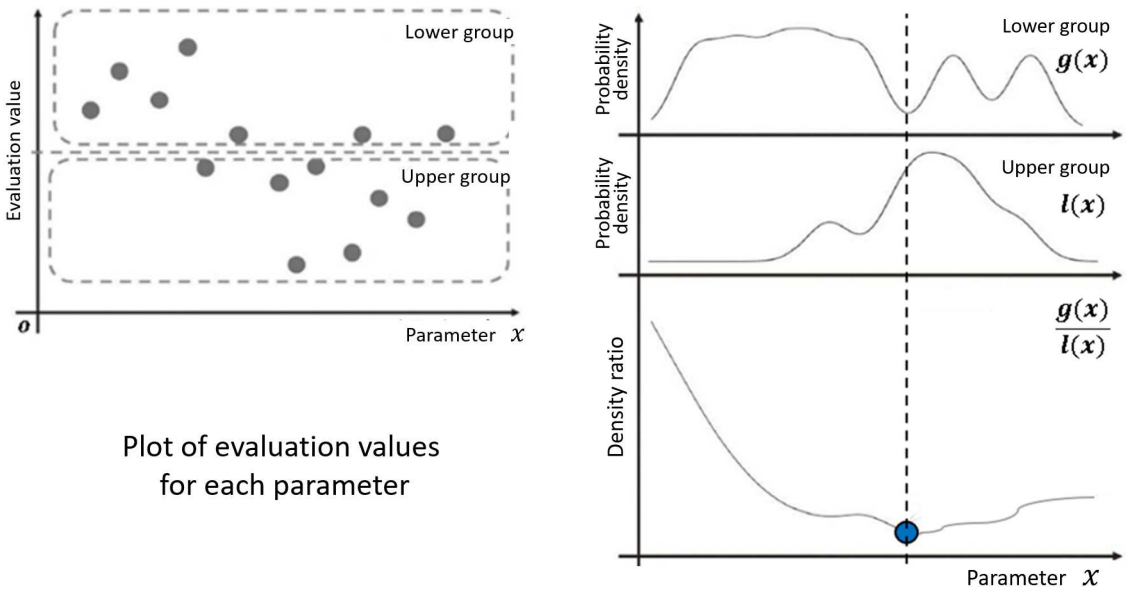
3.2. **Improved method of DQN.** We use an improved method of DQN, which incorporates two algorithms: Double DQN and Multi-step Learning, into DQN to improve the learning performance. Double DQN (DDQN) is a combination of DQN and Double Q-learning, which uses two networks in order to stabilize the learning performance during the update of the action-value function $Q(s, a)$. Multi-step Learning is an algorithm which improves the accuracy of action value estimation by learning from N-step rewards and action values at N-steps ahead. In this case, TD error $\delta_i(\theta_i)$ at state s_j is computed by the following equation:

$$\delta_i(\theta_i) = \sum_{n=0}^{N-1}\gamma^{n}r_{j+n} + \gamma^{N}\max_{a' \in \mathcal{A}} Q\left(s_{j+N}, a'; \theta_i^{-}\right) - Q(s_j, a_j; \theta_i) \tag{4}$$

4. **Parameter Optimization Using Bayesian Optimization.** Bayesian optimization is one of the promising methods in design of experiments for optimizing experimental parameters. In this research, we apply the Bayesian optimization framework Optuna [11] in order to optimize the network structure of DQN and the values of hyper-parameters.

Optuna is a method of design of experiments and it uses the TPE (Tree-structured Parzen Estimator) algorithm for Bayesian optimization. The principle of TPE is as follows. By applying Bayes' theorem to each distribution of parameter values that produced good results (the upper group) and those that produced bad results (the lower group), we can derive the parameter values which maximize the expected value of the evaluation function.

The behavior learning using DQN-based method is then performed using these derived parameter values as shown in Figure 2. After the behavior learning, the parameter values are updated to maximize the expected value of the evaluation function. Then the



(a) Division into two groups based on evaluation function value (minimization problem)

(b) Probability density

FIGURE 2. Principle of TPE in Bayesian optimization

behavior learning process is performed again using these updated parameter values. By repeating this process, Optuna can efficiently find the optimal parameter values. Grid search, which involves exploring all possible combinations of parameter values, is computationally expensive and time-consuming, whereas Bayesian optimization method Optuna can discover good parameter values in shorter amount of time.

5. **TurtleBot3 Burger Robot.** In this research, we use the ROBOTIS TurtleBot3 Burger robot [12] shown in Figure 3 as a mobile robot. TurtleBot3 Burger robot is a commercially available two-wheeled mobile robot which is equipped with LiDAR sensor and Raspberry Pi small computer. It is easy to install ROS (Robot Operating System) software. The size of TurtleBot3 Burger robot is $138 \times 178 \times 192$ [mm] and its weight is 1 [kg].

LiDAR is a 2D laser scanner which is capable of distance measurement in 360 [deg] around the device. TurtleBot3 Burger has ROBOTIS LDS-01 as LiDAR on the top of the robot as shown in Figure 3. The measuring range of LDS-01 is from 0.12 [m] to 3.5 [m] and its angle resolution is 1 [deg].

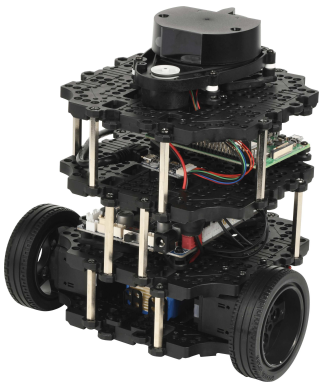


FIGURE 3. TurtleBot3 Burger

ROS (Robot Operating System) is one of the major open-source softwares for robot development. ROS provides a framework of data communication between the programs by distributed processing system. Moreover, it provides many kinds of software libraries and tools for robot development. By using ROS, it is possible to reduce the development time and cost to develop the robot software.

6. Problem Setting.

6.1. **Problem environment.** In this research, we assume the simulation environment shown in Figure 4, as the problem environment. The size of the problem environment is 1.8×1.8 [m] and it is surrounded by black walls with height of 25 [cm]. There are two goal areas and one obstacle with size of 0.4×0.4 [m] in the center.

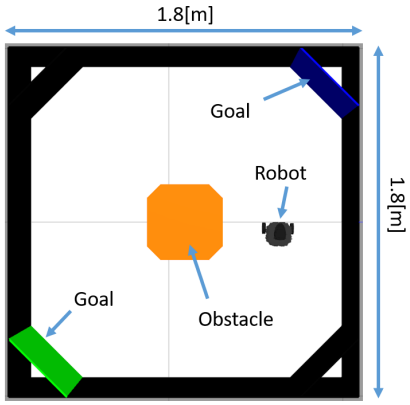


FIGURE 4. Problem environment

The observation information is a set of web camera image and 12 values of the LiDAR. We attached the web camera BUFFALO BSW200MBK to the front of the TurtleBot3 Burger robot with downward at 15 [deg] angle. It captures the field of view of 120 [deg] in front of the robot. It captures color images of 1920×1080 [pixel] and we reduce the image size into 48×27 [pixel]. Whereas, 12 values of the LiDAR are obtained from 360 [deg] range measurement at 30 [deg] intervals. The LiDAR used in this research is ROBOTIS LDS-01 as described in the previous section.

Using images from the web camera and the values from the LiDAR on the robot, we aim to realize that the mobile acquires the behavior to reach the goal area while avoiding the wall and obstacle.

6.2. **Definition of state, action and reward.** We implemented DQN-based method using PyTorch deep learning framework and reinforcement learning library PFRL [14] by Preferred Networks. As shown in Figure 1, the input values of DQN are state s which consists of a camera image and 12 values of LiDAR in 12 directions. All input values are normalized between 0 and 1. The camera image is an RGB image of size 48×27 [pixel], and the LiDAR values take a maximum value of 1 when the distance is 1.8 [m].

The input state s is decomposed into the camera image and LiDAR values. By convolution process and max-pooling process for the camera image, the feature maps are obtained. Then, the values from the LiDAR are combined with the flatted feature maps. Finally, the action values $Q(s, a)$ for all candidate action a are computed through the fully-connected layer.

The robot repeatedly selects one action from three candidate actions: a_1: go straight, a_2: turn left, a_3: turn right. One action step means one cycle from the observation of state s to the execution of the selected action. By executing the action a_1 (go straight), the robot moves forward by 30 [mm]. By executing the action a_2 (turn left) or a_3 (turn

right), the robot rotates by 0.28 [rad] counter-clockwise or clockwise while moving forward by 40 [mm].

The reward r_t is determined using the camera image and LiDAR values as follows:

$$r_t = \begin{cases} +1000 & \text{(in case the robot reaches the goal area)} \\ +1 & \text{(in case the robot is away from walls and obstacle)} \\ -1 & \text{(in case the robot is close to walls or obstacle)} \\ -100 & \text{(in case the robot collides to walls or obstacle)} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

6.3. **Restart method in case of collision.** During training, when the robot collides to walls or obstacle, the robot stops at once and the robot executes the restart action. The restart function is introduced that allows the robot to restart on its own, in order to save time and effort to return the robot to the starting position.

The restart method is illustrated in Figure 5. When the robot collides to walls or obstacle, it uses the LiDAR sensor to measure the distance between the robot and the surrounding walls or obstacle. The robot rotates until it finds the shortest distance in front of it, and then it moves backward. After that, the robot randomly rotates either in clockwise or in counter-clockwise by 90 [deg]. If the LiDAR sensor indicates that the distance from the surrounding walls or obstacle is more than 5 [cm], the restart is completed. Otherwise, the restart action is repeated.

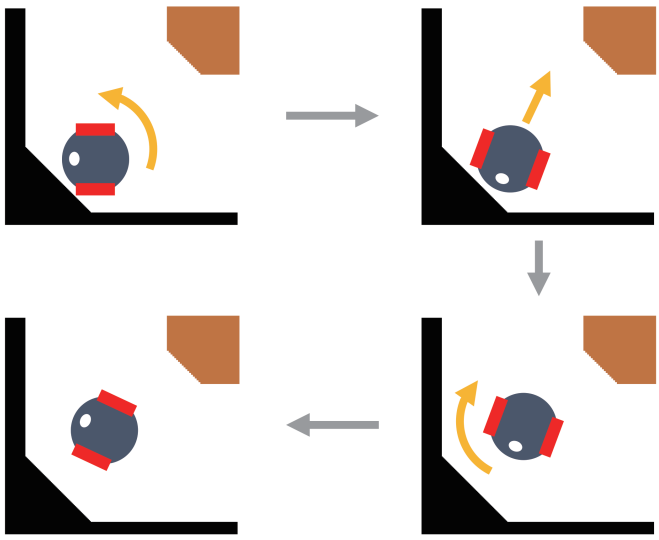


FIGURE 5. Restart method in case of collision

6.4. **Optimization targets of Bayesian optimization.** The candidate parameters to be optimized using the Bayesian optimization framework Optuna are shown below.

1) Combinations of convolutional layers and pooling layers in the network structure of DQN (2 patterns)
2) The number of hidden layers and the number of units in the hidden layers of fully-connected layers in the network structure of DQN (1∼3 layers)
3) Kind of the optimizer (Adagrad, Adadelta, Adam, RMSprop)
4) Batch size in mini-batch learning (8∼512)

7. **Experiment.** In the problem environment shown in Figure 4, the behavior learning by DQN-based method consists of 8,000 action steps, with 16 episodes where 1 episode which consists of 500 action steps. After learning, 5 evaluations (test runs) are conducted

starting from the position shown in Figure 6. The first 2 episodes (1,000 action steps) are for initial exploration, during which the robot does not learn and only collects the samples of experience. After 16 episodes, it is evaluated by averaging the rewards obtained from 5 evaluations (test runs) performed with the same hyper-parameter values. The values of hyper-parameters are updated using Optuna, and the behavior learning and the evaluation are performed again to approach the optimal values of the hyper-parameters.

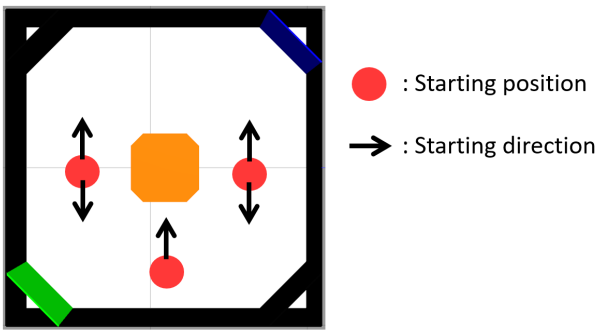


FIGURE 6. Starting position and direction in the test runs

We conducted 100 sets of experiments using Optuna described above in order to optimize the hyper-parameters of the DQN. As a result by using Optuna, we could get the combination of optimized hyper-parameters as shown below and the optimized network structure as shown in Figure 7.

1) Combinations of convolutional layers and pooling layers in the network structure of DQN (2 patterns)
   →Optimization result: Pattern 2 (Figure 7)
2) The number of hidden layers and the number of units in the hidden layers of fully-connected layers in the network structure of DQN (1∼3 layers)
   →Optimization result: 1 layer, 50 units
3) Kind of the optimizer (Adagrad, Adadelta, Adam, RMSprop)
   →Optimization result: Adagrad
4) Batch size in mini-batch learning (8∼512)
   →Optimization result: 32

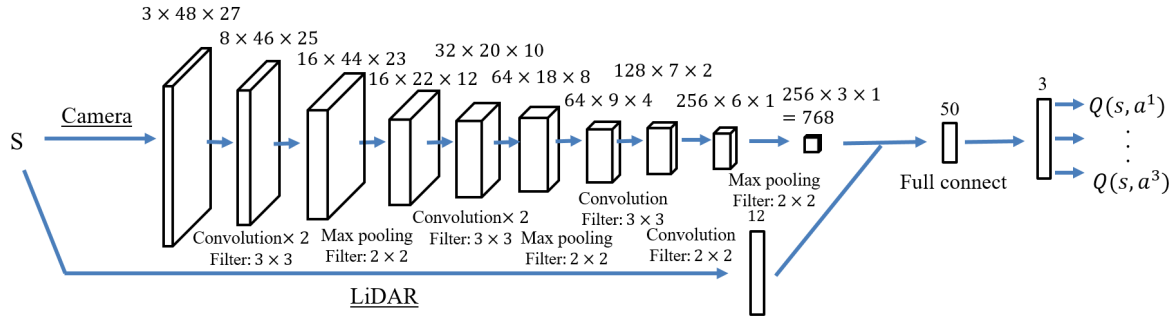


FIGURE 7. The network structure of DQN optimized by Optuna

Next, we conducted each 3 sets of simulation experiments by two optimization methods: one is using the hyper-parameters optimized by Optuna and the other is using the hyper-parameters manually optimized through preliminary experiments.

Table 1 shows the evaluation results of test run phase after the learning of 16 episodes (8000 action steps). From Table 1, we can see that the hyper-parameters optimized by Optuna enable the robot to reach the goal area with average of 4.00 times per 5 trials,

TABLE 1. Evaluation results by two optimization methods (after the learning of 16 episodes)

		Evaluation values (Mean)	Num of steps (Mean)	Num of goals (Sum)	Num of collisions (Sum)
Manual	1st	147.01	231.80	1.00	2.00
	2nd	457.97	388.80	2.00	0.00
	3rd	353.57	312.80	2.00	1.00
	Mean	**319.52**	**311.13**	**1.67**	**1.00**
Optuna	1st	873.51	26.20	4.00	1.00
	2nd	911.70	29.20	4.00	1.00
	3rd	873.01	40.00	4.00	1.00
	Mean	**886.07**	**31.80**	**4.00**	**1.00**
Manual→Optuna		**+566.55**	*−279.33*	**+2.33**	**±0.00**

while the hyper-parameters by manual optimization enable it to reach the goal area with average of only 1.67 times per 5 trials. The number of action steps to reach the goal area by Optuna optimization is much fewer (with −279.33 action steps) than that of manual optimization.

The mean rewards of 3 sets experiments obtained during the behavior learning by DQN-based method are illustrated in Figure 8. From Figure 8, we can see that using the hyper-parameters optimized by Optuna, the agent obtained higher reward than those by manual optimization after the 7th episode.

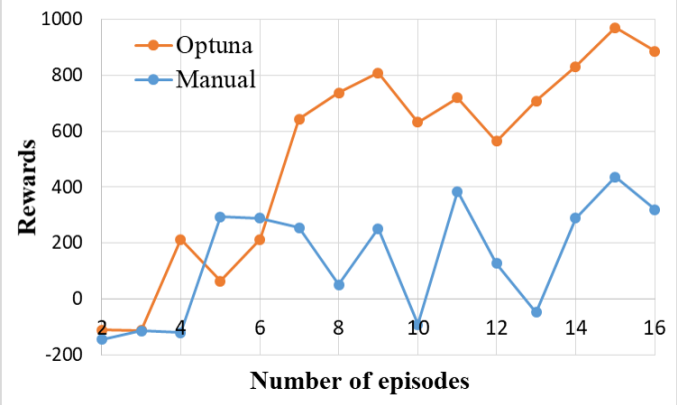


FIGURE 8. Change of the mean reward during behavior learning using the hyper-parameters by two optimization methods (mean of 3 sets experiments)

These results indicate that we can obtain high learning ability to reach the goal area avoiding collisions with walls and obstacle by optimizing hyper-parameters using the Bayesian optimization framework, Optuna.

8. **Conclusions.** In the simulation environment for the behavior acquisition problem of the mobile robot to reach the goal area avoiding walls and obstacle, we succeeded in optimizing both of the hyper-parameters and the network structure of the DQN, by using Bayesian optimization framework, Optuna. Furthermore, we confirmed that the hyper-parameters obtained by Optuna have higher learning ability than the empirically optimized hyper-parameters manually, through the simulation experiments.

As future works, it is neccesary to accelerate the optimization process of the hyper-parameters by Optuna in case the search space is quite large, because it takes long time

to search the hyper-parameters by Optuna. Based on the evaluation values of behavior learning using the hyper-parameters optimized by Optuna and the magnitude of the effect of each parameter which is also provided by Optuna, we think it is possible to determine the hyper-parameters that can be fixed in advance. Hence, it enables to reduce the processing time by Optuna by focusing the exploration of the hyper-parameters which have significant impact on the evaluation value. Moreover, we need to apply the obtained values of hyper-parameters using Optuna in the simulation environment to the real robot environment.

Acnowledgement. This research was supported by JSPS KAKENHI Grant Number JP19K12147 from the Japan Society for the Promotion of Science.

REFERENCES

[1] V. Mnih et al., Human-level control through deep reinforcement learning, *Nature*, vol.518, pp.529-533, 2015.
[2] D. Silver et al., Mastering the game of Go with deep neural networks and tree search, *Nature*, vol.529, pp.484-489, 2016.
[3] Y. LeCun et al., Gradient-based learning applied to document recognition, *Processdings of the IEEE*, vol.86, no.11, pp.2278-2324, 1998.
[4] R. Sota, A. Fukushima and T. Horiuchi, A study on improved method for behavior acquisition of mobile robot by deep reinforcement learning, *Proc. of 2022 Annual Conference on Electronics, Information and Systems, Institute of Electrical Engineers of Japan*, pp.1505-1506, 2022 (in Japanese).
[5] R. Kageyama et al., Train frontal obstacle detection method with camera-LiDAR fusion, *Quarterly Report of RTRI*, vol.63, no.3, pp.181-186, 2022.
[6] R. Gandhinathan et al., *ROS Robotics Projects*, 2nd Edition, Packt, 2019.
[7] I. Ekundayo, *OPTUNA Optimization Based CNN-LSTM Model for Predicting Electric Power Consumption*, Ph.D. Thesis, National College of Ireland, Dublin, 2020.
[8] A. Uemura et al., Stair recognition using CNN and step detection using depth camera, *Journal of Japan Society of Directories*, vol.21, no.1, pp.78-87, 2023 (in Japanese).
[9] S. Takaoka et al., Evaluation of hierarchical reinforcement learning methods using grid worlds, *Proc. of Game Programming Workshop 2019*, pp.172-176, 2019 (in Japanese).
[10] C. J. Watkins and P. Dayan, Technical note: Q-learning, *Machine Learning*, vol.8, pp.279-292, 1992.
[11] *https://www.preferred.jp/ja/projects/optuna/*, Accessed on November 20, 2023.
[12] *https://emanual.robotis.com/docs/en/platform/#turtlebot3*, Accessed on November 20, 2023.
[13] *https://ros.org*, Accessed on November 20, 2023.
[14] *https://github.com/pfnet/pfrl*, Accessed on November 20, 2023.